APPENDIX B

The attached Appendix is considered to be part of and included in the present Amendment.

```
YACC example
%token A
%token BODY_ UIBODY %token IMG
%token INPUT_ button %token INPUT _checkbox %token INPUT_ radio %token INPUT_ text
%token SELECT _UICOMBO %token SELECT _UILISTBOX %token SPAN
%token SPAN_ UILABEL %token TD
%token TD _UIBTN
%token TD _UIBTNTXT %token TD _UIMENUBAR %token TD UIMENUITEM
%token TD UITOOLBAR %token TD UIPANEL
%%
1. body
```
The symbol "body" is the starting symbol of the grammar and it refers to the BODY HTML tag. In fact the right hand side of the rule begins with the recognition of a BODY HTML tag, as the first DOM element that is analyzed is always this tag. Notice the use of error tokens throughout the rules denoting areas of the input that are ignored. The illustrated system takes advantage of the fact that the yacc generated LALR(1) parser uses an error handling and recovery mechanism that is suitable for ignoring certain parts of the input stream.
```
: BODY UIBODY error TD UIMENUBAR menu TD UITOOLBAR toolbar TD UIPANEL panel
```
The symbol stack of the parser either contains references to DOM elements in case of terminal symbols (tokens representing HTML tags, such as TD _UINHNUBAR above) or a pointer to the tree of generated UI objects in case of non-terminal symbols (such as menu, toolbar and panel above). The step below concatenates two trees of UI objects and puts the result in m_sResult, the output of the parser. Notice the fact that this action gets executed as a final step of the syntactical analysis, when this rule is reduced to the starting symbol.
Here, the tokens based on TD tags are to partition the input stream. The first TD tag with the `TD UIMENUBAR' classname attribute signals the beginning of the menu bar (menu). This is one example, where this grammar-based approach shows its strength: it is very easy to create rules that are applied locally (to a certain part of the input stream) and the context is always well defined and followed by the parser.
```
{
CParserSymbol *t;
m_sResult       JoinAsSiblings(t = JoinAsSiblings($4, $6), $8);
m_iResult = 1;
$$ = NULL;
}
|  error
{
m_sResult = NULL; m_iResult = 0;
$$ = NULL;
}
;
```
The following two groups of rules (menu, menu_item) recognize the menu (as shown on the top of the browser client area on the screenshot). The first rule group makes processing of several menu items possible. It is one way to express iteration of items in the grammar.
```
menu
: menu_item
{
$$ = $1;
}
| menu  menu_item
```
The created UI objects must be joined in a tree, that is the purpose of the JoinAsSiblings function.
```
   {
$$ = JoinAsSiblings($1, $2);
   }
;
menu_item
: TD UIMENUITEM
```
The constructor of the class CGenericObject is the place where the list of UI object

identifiers that correspond to the graphical elements displayed in the browser is generated. (See Figure 3, item 48). Inside the constructor the parser is able to access various properties of a DOM element, in this case its innerText and clientRects properties. In this simple instance there is a one-to-one relationship between the DOM element (the TD tag that is the only symbol on the right side of the rule - this element of the symbol stack is accessed with the $1) and the UI object.

```
{
            $$ = new CGenericObject(this, NULL, NULL, L"MENUITEM", L"ROLE_ SYSTEM
MENUITEM", 0, 0, 0, 0, -1, 1, $1);
}
| error
```

Anything other than the above TD_UIMENUITEM tags are ignored by this error alternative.

```
{
     $$ = NULL;
}
;
```

Iteration over toolbar buttons is done the same way as for menu items.

```
toolbar
: toolbar_item
{
      $$ = $1;
}
|    toolbar    toolbar_item
{
      $$ = JoinAsSiblings($1, $2);
}
;
```

2. Toolbar_item

The same logical UI abstraction (a button) might be implemented in many different ways even in one application. These alternatives must be present here in the grammar. These rules also present examples of logical UI objects that are created from more than one DOM element. The clientRects attribute is read from the SPAN tag while the logical name of the object might be found as the ALT attribute of the IMG tag (first alternative) or as innerText of the SPAN tag (second alternative).

```
: TD UIBTN ' { ' SPAN '{' IMG '}' '}'
     {
$$ = new CGenericObject(this, NULL, NULL, L"PUSHBUTTON", L"ROLE_SYSTEM_PUSHBUTTON",
-1, 0, 0, 0, -1, 2, $1, $5);
CComVariant vAlt;
CComBSTR sAlt("alt"); ((CPSHTMLElement*)$5)->m_pElement->getAttribute(sAlt, 0,
&vAlt);
if (vAlt.vt == VT _BSTR)
  ((CBXObjectTree*)$$)->m OD.odName =
::SysAllocString(vAlt.bstrVal);
}
| TD UIBTNTXT '{' SPAN '{' IMG 1}"}'
```

This alternative only differs from the Is` in the classname of the SPAN element. These classnames are mapped to different tokens although they can be unified in the scanner/tokenizes. This makes the parser more complex but makes the creation of UI objects simpler.

```
{
               $$ = new CGenericObject(this, NULL, NULL, L"PUSHBUTTON",
L"ROLE_SYSTEM_ PUSHBUTTON", 0, 0, 0, 0, -1, 2, $1, $5);
}
| error
{
             $$ = NULL;
}
;
```

Iteration over the rest of the controls is done the same way as for menu items.

3. panel

```
: panel_item
```

```
{
$$ = $1;
}
| panel panel_item
{
$$ = JoinAsSiblings($1, $2);
}
;
```
The following groups of rules describe the various controls that can be found in the example application.

4. Panel_item

```
: A
   {
$$ = new CGenericObject(this, NULL, NULL, L"LINK", L"ROLE SYSTEM LINK", 0, 0, 0, 0,
-1, 1, $1);
   | INPUT_button
   {
$$ = new CGenericObject(this, NULL, NULL, L"PUSHBUTTON", L"ROLE SYSTEM PUSHBUTTON",
-1, 0, 0, 0, -1, 1, $1); -
CComVariant vValue;
CComBSTR sValue("value"); ((CPSHTMLElement*)$1)->m_pElement->getAttribute(sValue, 0,
&vValue);
if (vValue.vt == VT_BSTR)
  ((CBXObjectTree*)$$)->m OD.odName = ::SysAllocString(vValue.bstrVai);
   }
   | INPUT checkbox SPAN UILABEL
   {
$$ = new CGenericObject(this, NULL, NULL, L"CHECKBUTTON", L"ROLE SYSTEM
CHECKBUTTON", 1, 0, 0, 0, -1, 2, $1, $2);
   }
   | INPUT radio SPAN UILABEL
   {
$$ = new CGenericObject(this, NULL, NULL, L"RADIOBUTTON", L"ROLE SYSTEM
RADIOBUTTON", 1, 0, 0, 0, -1, 2, $1, $2);
   }
panel_item_nl
   {
$$ = $1;
   }
TD ' { ' SPAN_UILABEL '{' I}! '}' TD If' panel_itemn1
```

This rule assigns labels to controls. panelitem_nl is a non-terminal which means that the symbol stack contains an already recognized control's object at the time of the execution of the action. The action itself copies the innerText of the second SPAN tag to the name member of the control's UI object.

```
   {
((CPSHTMLElement*) $3 )->m_pElement
>get_innerText ( SE ( ( CBXObj ectTree*) $9 ) - >m_OD . odName) ;
$$ = $9;
   }
   | error
```

This alternative makes the parser to ignore any unimportant input. Any sequence of tokens that are different from the previous alternatives are skipped.

```
   {
$$ = NULL;
   }
   ;
panel_item_n1
: INPUT -text
   {
$$ = new CGenericObject(this, NULL, NULL, L"TEXT",
L"ROLE SYSTEM TEXT", -1, 0, 0, 0, -1, 1, $1);
   }
   | SELECT_UICOMBO
```

```
   {
$$ = new CGenericObject(this, NULL, NULL, L"COMBOBOX", L"ROLE SYSTEM COMBOBOX", -1,
0, 0, 0, -1, 1, $1);
   }
   | SELECT_UILISTBOX
   {
$$ = new CGenericObject(this, NULL, NULL, L"LIST",
L"ROLE SYSTEM LIST", -1, 0, 0, 0, -1, 1, $1);
   }
   ;
   %%
C++ helper classes
typedef struct tagSBXObjectDescriptor
{
BSTR odClass;
BSTR odLogicalClass;
BSTR odName;
BSTR odLabel;
DWORD odStyle;
DWORD odStatus;
DWORD odFlags;
ULONG uRectCount;
[size _is(uRectCount)]
RECT *odRect;
BSTR odContents;
} SBXObjectDescriptor;
class CParserSymbol
{
public:
CParserSymbol();
virtual -CParserSymbol();
};
class CPSHTMLElement : public CParserSymbol
{
public:
CPSHTMLElement(const CHTMLElementPtr&, int iToken);
~CPSHTMLElement();
CHTMLElementPtr m_pElement;
int m_iToken;
};
class CBXObjectDescriptor : public SBXObjectDescriptor
{
public:
CBXObjectDescriptor();
~CBXObjectDescriptor();
SBXObjectDescriptor *pSBXObjectDescriptor);
};
class CBXObjectTree : public CParserSymbol
{
public:
CBXObjectTree(CBXObjectTree*, CBXObjectTree*);
~CBXObjectTree();
CBXObjectTree *m_pChild;
CBXObjectTree *m_pSibling;
CBXObjectDescriptor m_OD;
} ;
class CGenericObject : public CBXObjectTree
{
public:
CGenericObject(yyfparser*, CBXObjectTree*, CBXObjectTree*);
CGenericObject(yyfparser*, CBXObjectTree*, CBXObjectTree*, LPCWSTR szClass, LPCWSTR
szLogClass, LONG uName, DWORD dwStatus, LONG uStartRect, LONG uEndRect, LONG
uContents, ULONG uParamCount, ...);
```

```
-CGenericObject();
void FillRectArray(const CHTMLElement2Ptr &pElement2, const POINT &pt);
CElementArray m_aParams;
CDWordArray m_aParamTokens; LONG m_. uStartRect;
LONG m_ uEndRect;
LONG m_uContents;
CDWord2PtrMap m_mElementIndex;
void Construct(LPCWSTR szClass, LPCWSTR szLogClass, LONG uName, DWORD dwStatus, LONG
uStartRect, LONG uEndRect, LONG uContents, ULONG uParamCount, va_list) ;
} ;
CParserSymbol *JoinAsSiblings(CParserSymbol *&pl, CParserSymbol *&p2)
{
CBXObjectTree *pTreel = (CBXObjectTree*)pl;
if (pl && p2) {
    while (pTreel->m_pSibling) {
      pTreel = pTreel->m_pSibling;
}
pTreel->m_pSibling = (CBXObjectTree*)p2;
   }
   CParserSymbol *r = pl ? pl : p2;
   pl = NULL; p2 = NULL; return r;
   }
   CParserSymbol *JoinAsDescendants(CParserSymbol *&pl,
   CParserSymbol *&p2)
   {
CBXObjectTree *pTreel = (CBXObjectTree*)pl;
if (pl && p2) {
   while (pTreel->m_pChild) {
      pTreel = pTreel->m_pChild;
}
pTreel->m_pChild = (CBXObjectTree*)p2;
   }
   CParserSymbol *r = pl ? p1 : p2;
   pl = NULL;
   p2 = NULL;
   return r;
}
CParserSymbol::CParserSymbol() {
}
CParserSymbol::-CParserSymbol()
{
}
CPSHTMLElement::CPSHTMLElement(const CHTMLElementPtr &pElement, int iToken)
m_pElement(pElement), m_iToken(iToken)
{
}
CPSHTMLElement::-CPSHTMLElement()
{
}
CBXObjectDescriptor::CBXObjectDescriptor()
{
::ZeroMemory(this, sizeof(SBXObjectDescriptor));
}
CBXObjectDescriptor::-CBXObjectDescriptor()
{
if (odClass)
         ::SysFreeString (odClass);
if (odLogicalClass)
         ::SysFreeString(odLogicalClass);
if (odName)
         ::SysFreeString(odName);
if (odLabel)
         ::SysFreeString(odLabel);
```

```
if (odContents)
        ::SysFreeString(odContents); if (odRect)
delete odRect;
}
CBXObjectTree::CBXObjectTree(CBXObjectTree *pChild, CBXObjectTree *pSibling)
: m_pChild(pChild), mpSibling(pSibling)
{
}
CBXObjectTree::-CBXObjectTree()
{
if (m_pChild)
        delete m_pChild;
if (m pSibling)
        delete m_pSibling;
}
CGenericObject::CGenericObject(yyfparser *pParser, CBXObjectTree *pChild,
CBXObjectTree *pSibling)
: CBXObjectTree(pChild, pSibling), m_uStartRect(0), m_uEndRect(- 1), m_uContents(-1)
{
}
CGenericObject::CGenericObject(yyfparser *pParser, CBXObjectTree *pChild,
CBXObjectTree *pSibling, LPCWSTR szClass, LPCWSTR szLogClass, LONG uName, DWORD
dwStatus, LONG uStartRect, LONG uEndRect, LONG uContents, ULONG uParamCount, ...)
: CBXObjectTree(pChild, pSibling), m_uStartRect(0), m_uEndRect(- 1), m uContents(-1)
{
va_list argList;
va start (argList, uParamCount);
Construct(szClass, szLogClass, uName, dwStatus, uStartRect, uEndRect, uContents,
uParamCount, argList);
va end(argList);
}
CGenericObject::-CGenericObject()
{
}
void CGenericObject::Construct(LPCWSTR szClass, LPCWSTR szLogClass, LONG uName,
DWORD dwStatus, LONG uStartRect, LONG uEndRect, LONG uContents, ULONG uParamCount,
va_list argList)
{
   m_uStartRect = uStartRect;
    m_uEndRect = uEndRect;
   m_uContents = uContents;
   CParserSymbol *pElement;
   if (uParamCount)
   {
for (ULONG i = 0; i < uParamCount; i++)
{
        pElement = va arg(argList, CParserSymbol*);
        m_ aParams.pusTI _back(((CPSHTMLElement*)pElement)- >m_pElement);
        m_aParamTokens.push_back(((CPSHTMLElement*)pElement)-
>m iToken);
        CUnknownPtr pElementUnk(((CPSHTMLElement*)pElement)- >m_pElement);
m_ mElementIndex.insert(CDWord2PtrMap::value_type((DWORD)(IUnkn own*)pElementUnk,
(LPVOID)i));
        }
}
m_OD.odClass = ::SysAllocString(szClass);
m_OD.odLogicalClass = ::SysAllocString(szLogClass);
m_OD.odStatus = dwStatus;
if (uParamCount)
       {
        if (uName >=. 0) {
          m aParams[uName]->get innerText(&m OD.odName);
          CropWhitespace(m OD.oaName);
```

```
        }
}
if (m_OD.odRect)
    delete m_ D.odRect;
m_OD.odRect = NULL;
m_OD.uRectCount = 0;
for (LONG i = m_uStartRect; i <= m_uEndRect; i++)
         FillRectArray(CHTMLElement2Ptr(m_aParams[i]), ptOffset);
}
void CGenericObject::FillRectArray(const CHTMLElement2Ptr &pElement2, const POINT
&ptOffset)
{
if (!pElement2) {
return ;
   }
   CHTMLRectCollectionPtr pRectCollection;
pElement2->getClientRects(&pRectCollection);
   if (pRectCollection) {
ULONG uRectCount = m OD.uRectCount;
pRectCollection->get length((long*)&m_OD.uRectCount);
m_OD.uRectCount += uRectCount;
RECT *odRect = m_OD.odRect;
M_OD.odRect = new RECT[m_OD.uRectCount];
::ZeroMemory(m_OD.odRect, sizeof(RECT)*m_OD.uRectCount);
if (odRect) {
         ::CopyMemory(m_OD.odRect, odRect,
sizeof(RECT)*uRectCount);
delete odRect;
}
for (long i = uRectCount; i < (long)m_OD.uRectCount; i++)
    CComVariant idx = i - (long)uRectCount, rval;
    pRectCollection->item(&idx, &rval);
    if (rval.vt == VT_ DISPATCH) {
       CHTMLRectPtr pRect = rval.pdispVal;
        if (pRect) {
           pRect->get_left(&m_OD.odRect[i].left);
           pRect->get_top(&m_OD.odRect[i].top);
           pRect->get_right(&m_OD.odRect[i].right);
           pRect->get bottom(&m OD.odRect[i].bottom);
}
     }
   }
  }
}
```